MNIST Quick Overview:

Data: 70K handwritten digits (60K train, 10K test), each 28x28 grayscale

Model: Classifier neural network (10 output classes, one per digit. Model predicts probability of digit for each example.

Use Cases: Basic digit recognition (postal codes, checks, forms)

Why Used: Perfect for learning ML/testing ideas (small, clean dataset)

Limitation: Too simple for real applications (only digits, clean images)

In [88]:
```
from torchvision import datasets
from torchvision.transforms import ToTensor

# Download MNIST
train_data = datasets.MNIST(
    root = 'data',
    train = True,
    transform = ToTensor(),
    download = True
)

test_data = datasets.MNIST(
    root = 'data',
    train = False,
    transform = ToTensor()
)

# Convert to numpy arrays
x_train = train_data.data.numpy()
y_train = train_data.targets.numpy()
x_test = test_data.data.numpy()
y_test = test_data.targets.numpy()

# Reshape and normalize if needed
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

print(f"x_train shape: {x_train.shape}")  # Should be (60000, 28, 28)
print(f"x_test shape: {x_test.shape}")    # Should be (10000, 28, 28)
print(f"y_train shape: {y_train.shape}")  # Should be (60000,)
print(f"y_test shape: {y_test.shape}")    # Should be (10000,)
```

```
x_train shape: (60000, 28, 28)
x_test shape: (10000, 28, 28)
y_train shape: (60000,)
y_test shape: (10000,)
```

In [90]:
```
# Data prep for training

x_train_scaled = torch.FloatTensor(x_train)
x_test_scaled = torch.FloatTensor(x_test)

y_train = torch.FloatTensor(y_train.tolist()).long()
y_test = torch.FloatTensor(y_test.tolist()).long()
```

In [91]:
```
y_train
```

Out[91]:
```
tensor([5, 0, 4,  ..., 5, 6, 8])
```

In [81]:
```
import torch
import torch.nn as nn

class ClassificationNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ClassificationNet, self).__init__()
        # First flatten the 28x28 input to 784 (28*28)
        self.flatten = nn.Flatten()

        # Layer 1
        self.layer_1 = nn.Linear(784, 512)  # 784 = 28*28
        self.bn1 = nn.BatchNorm1d(512)

        # Layer 2
        self.layer_2 = nn.Linear(512, 256)
        self.bn2 = nn.BatchNorm1d(256)

        # Layer 3
        self.layer_3 = nn.Linear(256, num_classes)

        self.relu = nn.ReLU()
```

```
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        # Flatten the input
        x = self.flatten(x)

        # Forward pass through layers
        x = self.dropout(self.relu(self.bn1(self.layer_1(x))))
        x = self.dropout(self.relu(self.bn2(self.layer_2(x))))
        x = self.layer_3(x)
        return x
```

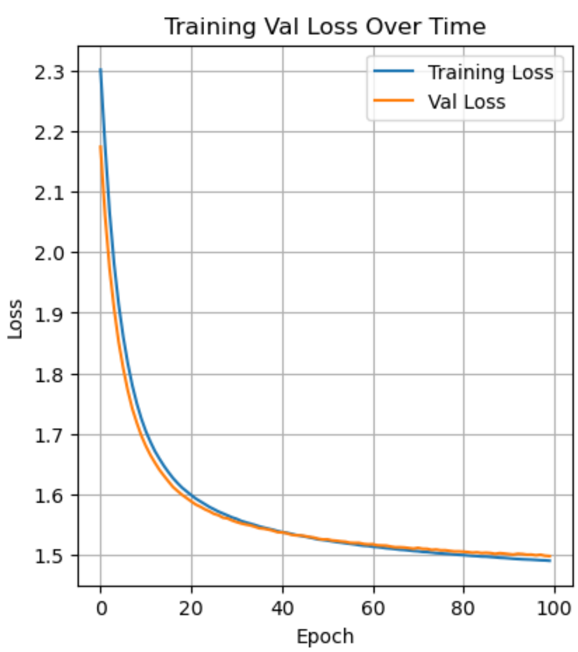
In [ ]:

```
# Model training

model = ClassificationNet()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)
num_epoch = 1000
train_loss_plot = []
loss_test = []
for epoch in range(num_epoch):
    model.train()
    outputs = model(x_train_scaled)
    loss = criterion(outputs,y_train)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Test accuracy
    train_loss_plot.append(loss.item())
    loss_test.append(criterion(model(x_test_scaled),y_test).item())
```

In [92]:

```
# Model training evaluate

import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(train_loss_plot, label='Training Loss')
plt.plot(loss_test, label='Val Loss')
plt.title('Training Val Loss Over Time')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.legend(loc='upper right')  # Specify location
```

Out[92]: <matplotlib.legend.Legend at 0x1b2a05550>



In [84]:

```
from sklearn.metrics import precision_recall_fscore_support, classification_report

# Evaluate on unseen test data
model.eval()
with torch.no_grad():
    # Get predictions
```

```python
    test_outputs = model(x_test_scaled)

    # Calculate loss
    test_loss = criterion(test_outputs, y_test)

    # Get predicted classes
    _, predicted = torch.max(test_outputs.data, 1)

    # Convert tensors to numpy for sklearn metrics
    y_true = y_test.cpu().numpy()
    y_pred = predicted.cpu().numpy()

    # Calculate accuracy
    total = y_test.size(0)
    correct = (predicted == y_test).sum().item()
    accuracy = 100 * correct / total

    # Calculate precision and recall for each class
    precision, recall, f1, _ = precision_recall_fscore_support(y_true, y_pred, average=None)

    # Print metrics
    print(f'Test Loss: {test_loss.item():.4f}')
    print(f'Test Accuracy: {accuracy:.2f}%')

    print("\nMetrics per class:")
    for i in range(len(precision)):
        print(f"\nClass {i}:")
        print(f"Precision: {precision[i]:.4f}")
        print(f"Recall: {recall[i]:.4f}")
        print(f"F1-score: {f1[i]:.4f}")

    # Print detailed classification report
    print("\nDetailed Classification Report:")
    print(classification_report(y_true, y_pred))
```

```
Test Loss: 1.4937
Test Accuracy: 97.52%

Metrics per class:

Class 0:
Precision: 0.9798
Recall: 0.9908
F1-score: 0.9853

Class 1:
Precision: 0.9826
Recall: 0.9930
F1-score: 0.9877

Class 2:
Precision: 0.9690
Recall: 0.9690
F1-score: 0.9690

Class 3:
Precision: 0.9696
Recall: 0.9782
F1-score: 0.9739

Class 4:
Precision: 0.9746
Recall: 0.9756
F1-score: 0.9751

Class 5:
Precision: 0.9786
Recall: 0.9753
F1-score: 0.9770

Class 6:
Precision: 0.9810
Recall: 0.9718
F1-score: 0.9764

Class 7:
Precision: 0.9680
Recall: 0.9698
F1-score: 0.9689

Class 8:
Precision: 0.9741
Recall: 0.9651
F1-score: 0.9696

Class 9:
Precision: 0.9749
Recall: 0.9613
F1-score: 0.9681

Detailed Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.98      0.99      0.99      1135
           2       0.97      0.97      0.97      1032
           3       0.97      0.98      0.97      1010
           4       0.97      0.98      0.98       982
           5       0.98      0.98      0.98       892
           6       0.98      0.97      0.98       958
           7       0.97      0.97      0.97      1028
           8       0.97      0.97      0.97       974
           9       0.97      0.96      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.97      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

In [1]:

```python
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# Create sample data
np.random.seed(42)
n_samples = 300
```

```python
# Generate feature data
X = np.random.normal(size=(n_samples, 2))

# Generate target variable (3 classes)
# Class 0: Low values of both features
# Class 1: High values of first feature
# Class 2: High values of second feature
y = np.zeros(n_samples)
y[X[:, 0] > 0.5] = 1
y[X[:, 1] > 1.0] = 2

# Convert to DataFrame for better visualization
df = pd.DataFrame(X, columns=['Feature_1', 'Feature_2'])
df['Target'] = y

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Print coefficients for each class
print("\nModel Coefficients:")
for i, class_label in enumerate(model.classes_):
    print(f"\nClass {class_label}:")
    print(f"Feature 1: {model.coef_[i][0]:.4f}")
    print(f"Feature 2: {model.coef_[i][1]:.4f}")
    print(f"Intercept: {model.intercept_[i]:.4f}")

# Example prediction for new data
new_data = np.array([[0.5, 1.2]])
new_data_scaled = scaler.transform(new_data)
prediction = model.predict(new_data_scaled)
probabilities = model.predict_proba(new_data_scaled)

print("\nPrediction for new data point [0.5, 1.2]:")
print(f"Predicted class: {prediction[0]}")
print("Probabilities for each class:")
for i, prob in enumerate(probabilities[0]):
    print(f"Class {i}: {prob:.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.95      0.97      0.96        37
         1.0       0.93      0.93      0.93        15
         2.0       0.86      0.75      0.80         8

    accuracy                           0.93        60
   macro avg       0.91      0.89      0.90        60
weighted avg       0.93      0.93      0.93        60


Model Coefficients:

Class 0.0:
Feature 1: -1.9889
Feature 2: -1.7381
Intercept: 2.7600

Class 1.0:
Feature 1: 2.5937
Feature 2: -1.6915
Intercept: -0.1179

Class 2.0:
Feature 1: -0.6048
Feature 2: 3.4297
Intercept: -2.6422

Prediction for new data point [0.5, 1.2]:
Predicted class: 2.0
Probabilities for each class:
Class 0: 0.1200
Class 1: 0.0792
Class 2: 0.8008
```

In [5]:

```
probabilities
```

Out[5]:

```
array([[0.11999349, 0.07923127, 0.80077524]])
```

In [ ]:

```

```